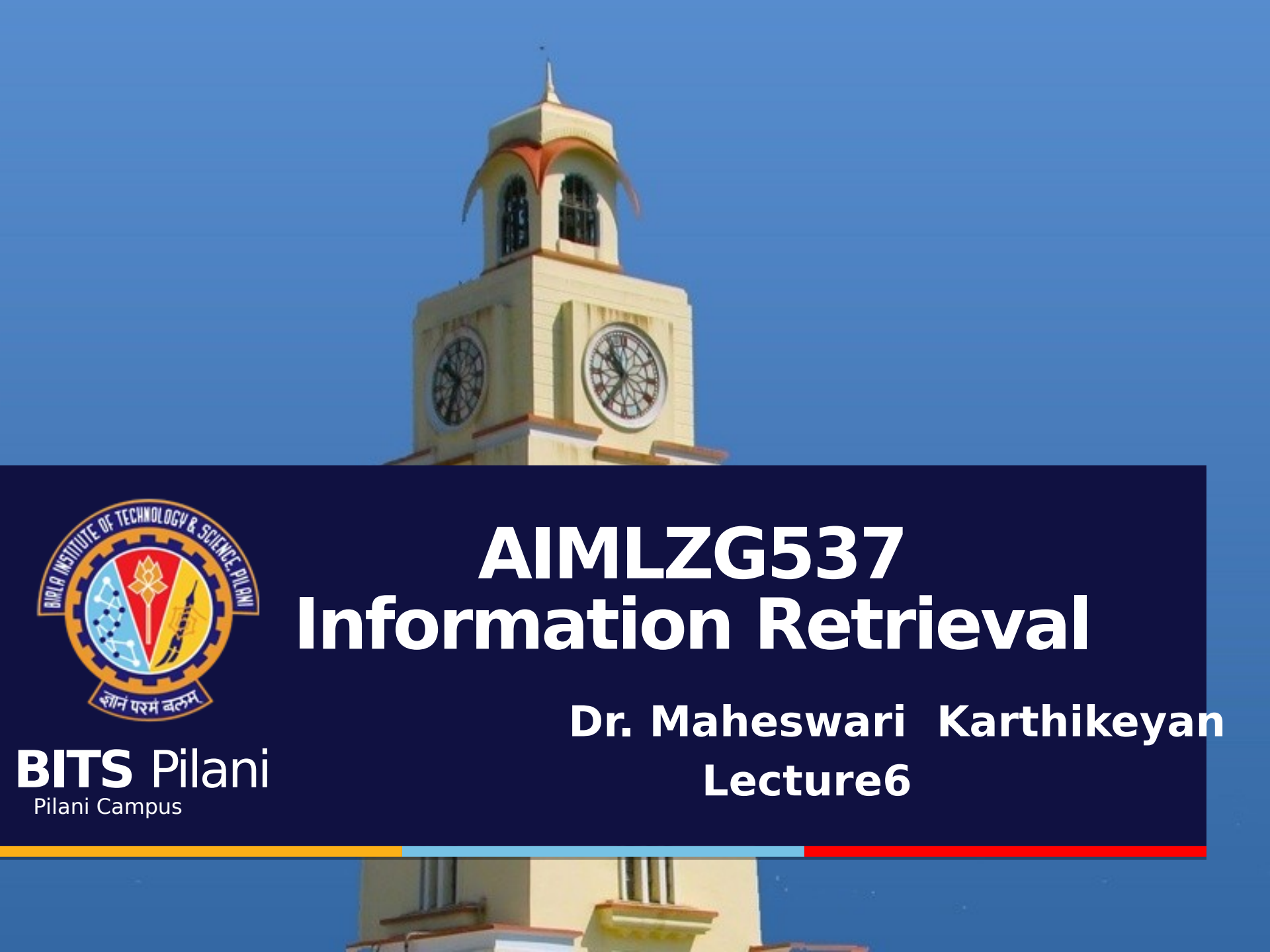


AIMLZG537
Information Retrieval

Dr. Maheswari Karthikeyan

Lecture6

BITS Pilani
Pilani Campus

Chapter 6
Vector Space Model

Slides are adapted from Introduction to Information Retrieval;Manning,Raghavan
http://nlp.stanford.edu/IR-book

Vector Space Model

innovate achieve lead

- Term frequency and weighting
- The vector space model for scoring

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
- Not good for the majority of users.
  - Most users are incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
  - This is particularly true of web search.

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- When a system produces a ranked result set,
  - The size of the result set is not an issue
  - Just show the top k ($\approx$ 10) results
  - Don't overwhelm the user

Ranked retrieval models

- Return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query “match”.

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Jaccard coefficient

- A commonly used measure of overlap of two sets A and B
- jaccard$(A,B) = |A \cap B| / |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient

$$\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$$

- Larger the term overlap of the two sets *A* and *B,* the better.
- Larger the *proportion* of term overlap with respect to the sizes of the sets A and B, the better.

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?

Query: *ides of march*

Document 1: *caesar died in march*

Document 2: *the long march march*

Document 3: *the long march end april*

Issues with Jaccard for scoring

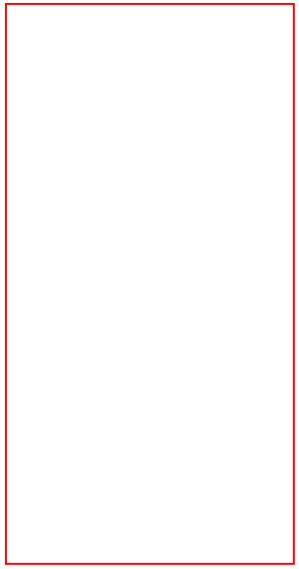
- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing the length of document

Term-document incidence matrix

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector



Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at "recovering" positional information later.
- For now: bag of words model

Term frequency - tf

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores.
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Example

- Compute the **Jaccard** matching score and the **tf** matching score for the following query-document pairs.

a) q: [information on cars] d: “all you have ever wanted to know about cars”

b) q: [information on cars] d: “information on trucks, information on planes, information on trains”

c) q: [red cars and trucks] d: “cops stop red cars more often”

Log-frequency weighting

- The log frequency weight of term **t in d** is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\mathrm{tf}_{t,d} \rightarrow w_{t,d}$:
  0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

- Score for a document-query pair: sum over terms *t* in both *q* and *d*:

$$\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

Document Frequency

- The document frequency is the **number of documents in the collection that the term t occurs in.**
- Use the frequency of the term in the collection for weighting and ranking.

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.

Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

Document Frequency

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection
- (e.g., **arachnocentric)**.
- A document containing this term is very likely to be relevant.
- We want high weights for rare terms like **arachnocentric**

Document Frequency

- Frequent terms are less informative than rare terms
- Consider a term in the query that is frequent in the collection (e.g., good, increase, line)
- A document containing this term is more likely to be relevant than a document that doesn't
- But words like good, increase and line are not sure indicators of relevance
- For frequent terms like good, increase, and line, we want positive weights, but lower weights than the rare terms.

Document Frequency

- Need high weights for rare terms like arachnocentric
- Need low (positive) weights for frequent words like good, increase, and line
- Can use the document frequency to factor this into computing the matching score.

idf weight

- df_t is the document frequency, the number of documents that **t** occurs in.
- df_t is an inverse measure of the informativeness of term t.
- We define the **idf** weight of term t as follows:
  - **$i\ df_t = \log (N / df_t)$**

(N is the number of documents in the collection.)

- idf_t is a measure of the informativeness of the term.

idf example

suppose N = 1,000,000

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
  - iPhone
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

tf-idf weighting

- The tf-idf weight of a term is the product of its **tf** weight and its **idf** weight.

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign.
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document (term frequency)
- Increases with the rarity of the term in the collection (inverse document frequency)

Example

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

Exercise 6.10
Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in Figure 6.9. Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values from Figure 6.8.

tf – idf for (cars, doc1) = (1+log27) * 1.65 = 2.43 * 1.65 = 4

tf – idf for (cars, doc2) = (1+log 4) * 1.65 = 2.64

tf – idf for (cars, doc2) = (1+log 24) * 1.65 = 3.93

tf – idf score for term cars = 4+2.64+3.93= 10.57

Similar manner the scores for other terms has to be computed.

Example - Tf-idf scores

Tf-idf computation without taking logarithmic tf (raw tf value)

terms	Doc1	Doc2	Doc3
Car	44.55	6.6	39.6
Auto	6.24	68.64	0
Insurance	0	53.46	46.98
Best	21	0	25.5

Score for a document given a query

Binary → count → weight matrix

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Documents as Vectors

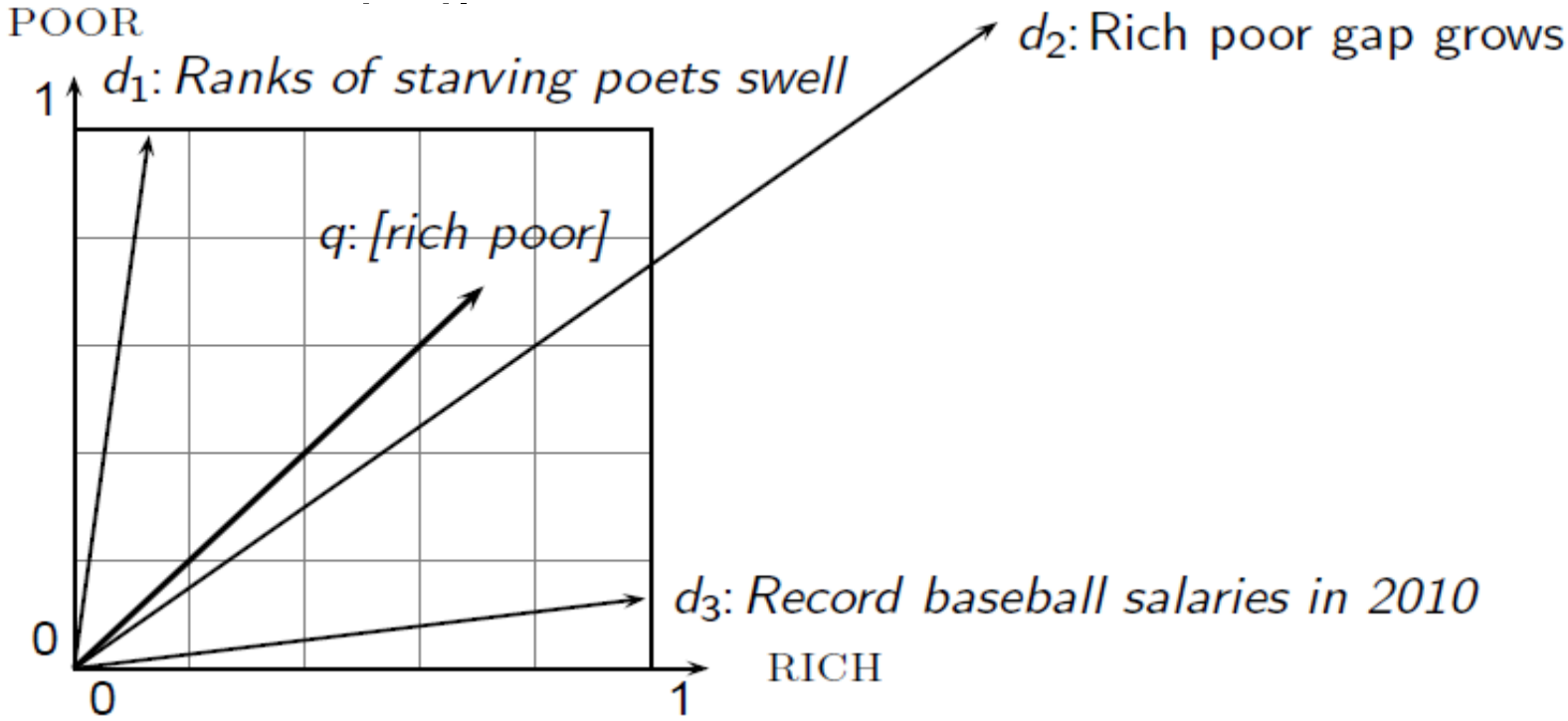
- Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$
- It is a |V|-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

Queries as vectors

- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- How to quantify the similarity between the vectors?

Similarity using distance(difference)

- The Euclidean distance between q and d_2 is large even though the distribution of terms in the query q and the distribution of terms in the document d_2



Use angle instead of distance

- Take a document *d* and append it to itself. Call this document *d′*.
- "Semantically" d and d′ have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.

- **Key idea: Rank documents according to angle with query.**

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in increasing order of the angle between query and document
  - Rank documents in decreasing order of cosine angle between query and document
  - Cosine is a monotonically decreasing function for the interval [0°, 180°]

Cosine(query, document)

Dot product

q_i is the tf-idf weight of term i in the query

d_i is the tf-idf weight of term i in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ …

or,

equivalently, the cosine of the angle between

Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

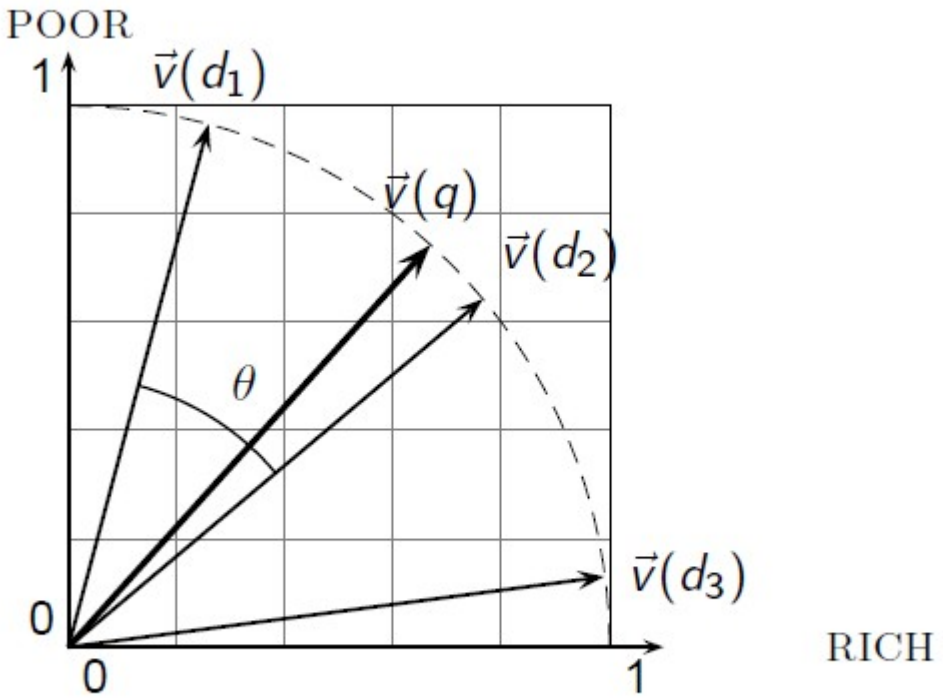
- for q, d length-normalized.

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

- Dividing a vector by its L_2 norm makes it a unit (length) vector.
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
    - Long and short documents now have comparable weights

Cosine similarity



Cosine similarity among 3 documents

How similar are the novels?

SaS: Sense and Sensibility , PaP: Pride and Prejudice, and

WH: Wuthering Heights

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

Cosine similarity among 3 documents

innovate achieve lead

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈

0.789 × 0.832 + 0.515 × 0.555 + 0.335 × 0.0 + 0.0 × 0.0

≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

3.02^2 + 2^2 + 1.30^2
= SQRT(15.05) = 3.87

3.06/3.87 = 0.79

Computing cosine scores

CosineScore(q)

```
1   float Scores[N] = 0
2   float Length[N]
3   for each query term t
4   do calculate w_{t,q} and fetch postings list for t
5       for each pair(d, tf_{t,d}) in postings list
6       do Scores[d] += w_{t,d} × w_{t,q}
7   Read the array Length
8   for each d
9   do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

tf-idf weighting variants

Term frequency		Document frequency		Normalization	
n (natural)	$tf_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-df_t}{df_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(tf_{t,d})}{1+\log(ave_{t \in d}(tf_{t,d}))}$				

Weighting may differ in queries vs. documents

- We often use different weightings for queries and documents.
- Notation: ddd.qqq
- Example: lnc.ltn
- document: logarithmic tf, no df weighting, cosine normalization
- query: logarithmic tf, idf, no normalization
- **Example query: “best car insurance”**
- **Example document: “car insurance auto insurance**”

Example (Inc.ltn)

innovate achieve lead

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto										
best										
car										
insurance										

Example

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**					**1**				
best	**1**					**0**				
car	**1**					**1**				
insurance	**1**					**2**				

Example

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**				**1**	**1**			
best	**1**	**1**				**0**	**0**			
car	**1**	**1**				**1**	**1**			
insurance	**1**	**1**				**2**	**1.3**			

Example

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**			**1**	**1**			
best	**1**	**1**	**50000**			**0**	**0**			
car	**1**	**1**	**10000**			**1**	**1**			
insurance	**1**	**1**	**1000**			**2**	**1.3**			

Example

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**	**2.3**		**1**	**1**			
best	**1**	**1**	**50000**	**1.3**		**0**	**0**			
car	**1**	**1**	**10000**	**2.0**		**1**	**1**			
insurance	**1**	**1**	**1000**	**3.0**		**2**	**1.3**			

Example

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**	**2.3**	**0**	**1**	**1**			
best	**1**	**1**	**50000**	**1.3**	**1.3**	**0**	**0**			
car	**1**	**1**	**10000**	**2.0**	**2.0**	**1**	**1**			
insurance	**1**	**1**	**1000**	**3.0**	**3.0**	**2**	**1.3**			

Example

Doc length =

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**	**2.3**	**0**	**1**	**1**	**1**		
best	**1**	**1**	**50000**	**1.3**	**1.3**	**0**	**0**	**0**		
car	**1**	**1**	**10000**	**2.0**	**2.0**	**1**	**1**	**1**		
insurance	**1**	**1**	**1000**	**3.0**	**3.0**	**2**	**1.3**	**1.3**		

Example

Doc length =

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**	**2.3**	**0**	**1**	**1**	**1**	**0.52**	
best	**1**	**1**	**50000**	**1.3**	**1.3**	**0**	**0**	**0**	**0**	
car	**1**	**1**	**10000**	**2.0**	**2.0**	**1**	**1**	**1**	**0.52**	
insurance	**1**	**1**	**1000**	**3.0**	**3.0**	**2**	**1.3**	**1.3**	**0.68**	

Example

Doc length =

Query **Document**

Word	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**	**2.3**	**0**	**1**	**1**	**1**	**0.52**	**0**
best	**1**	**1**	**50000**	**1.3**	**1.3**	**0**	**0**	**0**	**0**	**0**
car	**1**	**1**	**10000**	**2.0**	**2.0**	**1**	**1**	**1**	**0.52**	**1.04**
insurance	**1**	**1**	**1000**	**3.0**	**3.0**	**2**	**1.3**	**1.3**	**0.68**	**2.04**

Final similarity score between query and document = 3.08

Example

- Reconstruct the Table in the previous slides with the weighting as **lnc.ltc**

Word	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	product
auto	**0**	**0**	**5000**	**2.3**	**0**	**0**	**1**	**1**	**1**	**0.52**	**0**
best	**1**	**1**	**50000**	**1.3**	**1.3**	**0.34**	**0**	**0**	**0**	**0**	**0**
car	**1**	**1**	**10000**	**2.0**	**2.0**	**0.52**	**1**	**1**	**1**	**0.52**	**0.27**
insurance	**1**	**1**	**1000**	**3.0**	**3.0**	**0.78**	**2**	**1.3**	**1.3**	**0.68**	**0.53**

Exercise 6.19

- Compute the vector space similarity between the query **“digital cameras”** and the document **“digital cameras and video cameras”** by filling out the empty columns in Table 6.1. Assume N = 10,000,000, logarithmic term weighting for query and document, idf weighting for the query only and cosine normalization for the document only. Treat and as a stop word. Enter term counts in the tf columns. What is the final similarity score?

	query					document			
word	tf	wf	df	idf	q_i = wf-idf	tf	wf	d_i = normalized wf	$q_i \cdot d_i$
digital			10,000						
video			100,000						
cameras			50,000						

▶ **Table 6.1** Cosine computation for Exercise 6.19.

Exercise 6.19 – Solution

Word	Query					document			qi*di
	tf	wf	df	idf	qi=wf-idf	tf	wf	di=normalized wf	
digital	1	1	10,000	3	3	1	1	0.52	1.56
video	0	0	100,000	2	0	1	1	0.52	0
Cameras	1	1	50,000	2.3	2.3	2	1.3	0.68	1.56

Similarity score = 1.56+1.56 = 3.12

Summary – vector space ranking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top K (e.g., $K = 10$) to the user

Problems with Vector Space Model

- Missing semantic information (e.g. word sense).
- Missing syntactic information (e.g. phrase structure,

  word order, proximity information).
- Assumption of term independence (e.g. ignores synonymy).
- Lacks the control of a Boolean model (e.g., requiring a term to appear in a document).
- Given a two-term query “A B”, may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.

References

- Ranked Retrieval

https://www.youtube.com/watch?v=CTwqDj5gtLo 5

- Jaccard Score (14 mins)

https://www.youtube.com/watch?v=eqpzVedX1kY&t=605s

- TF and bag of words (24 mins)

https://www.youtube.com/watch?v=n_ul9GIOIkQ

- WDM 60: Inverse Document Frequency (16 mins)

https://www.youtube.com/watch?v=97Ey6aCeask

- WDM 61: TF-IDF Score (16 mins)

https://www.youtube.com/watch?v=r8zeOv638-A

- WDM 62: Documents AS TF-IDF Vectors (16 mins)

https://www.youtube.com/watch?v=zUyWj7Bx8HM

- WDM 63: Length Normalization (28 mins)

https://www.youtube.com/watch?v=70n2xxgEk7w

- Cosine Similarity Example (24 mins)
- https://www.youtube.com/watch?v=HW9W6EBytLg WDM 66: Variants of TF IDF Weights - YouTube